Learning User Preferences via Reinforcement Learning with Spatial Interface Valuing

Miguel Alonso Jr.[(✉)]

School of Computing and Information Sciences,
Florida International University, Miami, FL 33199, USA
malonsoj@cs.fiu.edu
http://dsail.fiu.edu

Abstract. Interactive Machine Learning is concerned with creating systems that operate in environments alongside humans to achieve a task. A typical use is to extend or amplify the capabilities of a human in cognitive or physical ways, requiring the machine to adapt to the users' intentions and preferences. Often, this takes the form of a human operator providing some type of feedback to the user, which can be explicit feedback, implicit feedback, or a combination of both. Explicit feedback, such as through a mouse click, carries a high cognitive load. The focus of this study is to extend the current state of the art in interactive machine learning by demonstrating that agents can learn a human user's behavior and adapt to preferences with a reduced amount of explicit human feedback in a mixed feedback setting. The learning agent perceives a value of its own behavior from hand gestures given via a spatial interface. This feedback mechanism is termed Spatial Interface Valuing. This method is evaluated experimentally in a simulated environment for a grasping task using a robotic arm with variable grip settings. Preliminary results indicate that learning agents using spatial interface valuing can learn a value function mapping spatial gestures to expected future rewards much more quickly as compared to those same agents just receiving explicit feedback, demonstrating that an agent perceiving feedback from a human user via a spatial interface can serve as an effective complement to existing approaches.

Keywords: Human computer interaction ·
Interactive machine learning · Reinforcement learning ·
Artificial intelligence

1 Introduction

Reinforcement Learning (RL) is an area of machine learning that seeks to create agents that learn what to do in uncertain, stochastic environments. That is,

RL agents learn how to map situations to actions with the goal of maximizing some type of reward [24]. A core tenant of RL that is touted as a key benefit is that agents learn to do this autonomously, without human intervention, in an unsupervised fashion. The typical engineering workflow in the design of RL agents is (1) a reward function is designed, (2) input and output channels are selected, and (3) an RL learning algorithm is designed [2]. And once the RL pipeline has been designed and implemented, little if any human intervention is needed during the learning process.

However, for real-world situations, where the high complexity of the world cannot easily be modeled in simulation environments, autonomous learning is often not feasible, primarily because of three key issues: the difficulty in specifying reward functions [20], the constraints on exploration due to the presence of potentially catastrophic outcomes for agents and humans alike [9], and difficulty in communicating goals to RL agent so as to avoid being misinterpreted [3]. As a result, many researchers have proposed introducing a human in the loop of training RL agents, to varying degrees of success [1,2,12,13,17,18].

These examples not only have elements of machine learning, but also elements of human-computer interaction (HCI). First coined in 1983 by Card et al. [5,6], HCI as a discipline is concerned with the design, evaluation and implementation of interactive computing systems for human use and with the study of major phenomena surrounding them [10]. Similarly, there has been some work related to having RL agents learn user preferences [7,26]. For example, Veeriah et al. [26] demonstrated that an RL agent can learn a human user's intent and preferences via ongoing interactions where the human user provides feedback through facial expressions, as well as, through explicit negative rewards. Collectively, the interface between machine learning and human computer interaction is referred to as Interactive Machine Learning (iML). In this paper, I explore combining Reinforcement Learning techniques with spatial interfaces to create agents that learn a value function that relates a user's body language, specifically a thumbs up or thumbs down gesture, to expectations of future rewards.

1.1   Interactive Machine Learning

Interactive Machine Learning (iML) is a subfield of Machine Learning that creates systems that operate in environments alongside human operators, where the human and the machine collaborate to achieve a task and both the human and ML system can be agents in the environment [11]. Many times, iML is used to extend or amplify the capabilities of a human in cognitive or physical ways. For this to be successful, the machine must adapt to the users' intentions and preferences, learning about their human counterparts' behavior. In much of the current research in iML, a user must convey feedback in some way to the iML agent.

One popular method is directly, such as through the click of a mouse, or through spoken word. This is known as explicit human feedback. Implicit human feedback on the other hand, is a mechanism through which a human can guide an iML agent's learning process through subtle cues, such as body language.

And yet a third type of feedback is known as mixed human feedback which combines explicit and implicit feedback [4]. Although all three forms of feedback impose some cognitive load on the human, explicit feedback carries the heaviest cognitive burden, especially in real-world settings where humans have additional cognitive loads due to the environment. The main objective of this work is to extend the current state of the art in iML by demonstrating that a learning agent can learn a human user's behavior and adapt to the human's preferences with a reduced amount of explicit human feedback in a mixed feedback setting.

1.2 Related Work

With the growth of the machine learning field over the last decade, there has been much effort in the community to create successful interactions between humans and machine learning systems with the goal of increase performance across a myriad of tasks. Many approaches to human-in-the-loop machine learning have focused primarily on agents learning from humans via explicit rewards. For example, Thomaz and Breazel [25] used a simulated RL robot and had a human teach the robot in real-time to perform a new task. They presented three interesting findings: (1) humans use the reward channel for feedback, as well as for future-directed guidance; (2) humans exhibited a positive bias to their feedback using the signal as a motivational channel; and (3) humans change their behavior as they develop a mental model of the robotic learner.

More recently, Knox & Stone [14,15] introduced the TAMER framework (Training an Agent Manually via Evaluative Reinforcement). TAMER's system for learning from human reward is novel in three key ways:

– TAMER addresses delays in human evaluation through credit assignment
– TAMER learns a predictive model of human reward
– and at each time step, TAMER chooses the action that is predicted to directly elicit the most reward, disregarding consideration of the action's effect on future state (i.e., in reinforcement learning terms, TAMER myopically values state-action pairs using a discount factor of 0).

According to Knox & Stone, "TAMER is built from the intuition that human trainers can give feedback that constitutes a complete judgement on the long-term desirability of recent behavior." One drawback of this method, however, is that when the user needs to modify the agent's behavior, the model would need to be changed, for example, via additional rewards from the user.

Another interesting approach is by Christiano et al. [7]. Here, the agent is trained from a neural network known as the 'reward predictor', instead of the classical RL approach of using the rewards it collects as it explores an environment. There are three processes running in parallel:

– A RL agent explores and interacts with its environment
– Periodically, a pair of 1–2 s clips of its behavior is sent to a human who is then asked to select the best one indicating steps toward fulfilling the desired goal

– The human's choice is used to train a reward predictor, which is then used
to train the agent

They showed that overtime, the agent learns to maximize the reward from the
predictor and improve its behavior according to the human's preferences. How-
ever, this methodology introduces substantial lag between the human feedback
and the agent's learning.

Another approach by Veeriah et al. [26] uses body language as one of the
drivers of learning. In contrast to the other approaches mentioned, this approach
is concerned with designing a general, scalable agent that would allow a human
user to change the agent's behavior according to preferences with minimal human
feedback. In this work, they used facial expressions to provide the feedback, not
as a channel for control, but rather as a means of valuing the agent's actions.

This work is an extension of the work by Veeriah et al. [26] to hand gestures
captured by a 3D spatial interface.

1.3   Problem Statement

There are many domains for which human and computing systems, enabled by
some form of machine learning, work collaboratively to achieve various tasks.
Once such domain of human computer interaction is in the realm of prosthe-
ses, whereby a human operator controls an electronic prosthetic limb [8,16].
Grip selection, for example, is one of the main tasks a human operating a pros-
thetic limb performs. Performing grasping of common objects with the prosthesis
requires choosing the correct grip from an array of various configurations. Most
systems cycle through grips via some feedback mechanism, typically a roll or
pronation/suprenation [21].

Thus, a grip selection task environment was created in order to evaluate
the machine learning system. This problem is taken from the real-world task of
selecting an appropriate grip pattern for grasping a given object by a user that
is operating a prosthetic arm, as mentioned above. Typically for most prosthetic
arms, there are a set of n discrete grips and depending on the type of object or
situation, the correct grip is defined according to the user's preference. These
preferences are normally setup by a clinician and must be periodically revisited
as the user gains experience with the prosthesis [21]. For this experiment, the
agent must select the correct grip for a random object presented, move forward,
and grasp the object. The episode ends when the agent successfully grasps the
object. In this way, the agent learns the user's gripping preferences in an ongoing
and online fashion, reducing the need for multiple clinical visits.

2   Reinforcement Learning Algorithm Background

Reinforcement Learning is a branch of Machine Learning that allows agents
to decide what to do, that is, how to map situations to actions with the goal
of maximizing a numerical reward signal. RL agents typically interact with an

environment, either real-world or simulated, gaining experience with each inter-
action, and improving their performance, as measured by the reward signal.
Mathematically, Markov Decision Processes (MDPs) are used to describe RL
problems. There are several algorithms that solve MDPs, many in optimal ways.
The RL learning algorithm that was investigated for the grip selection task was
SARSA(λ), which is an on-policy temporal difference (TD) control algorithm for
MDPs [24].

2.1   Markov Decision Processes

A Markov Decision Process is a tuple $(S, A, P_s a, \gamma, R)$, where:

- S is a set of *states*, $S \in \{s_0, s_1, \ldots s_m\}$
- A is a set of *actions*, $A \in \{a_0, a_1, \ldots a_m\}$
- P_{sa} are the state *transition probabilities*
- $\gamma \in [0, 1)$ is the *discount factor*, and controls the influence of future rewards
  in the present
- $R : S \times A \mapsto \mathbb{R}$ is the *reward function*, which returns a real value every time
  the agent moves from one state to the other due to an action

For each state $s \in S$ and action $a \in A$, P_{sa} is a probability distribution over the
state space and gives the distribution over what states the system will transition
to if action a is taken when the system is in state s. Starting from an initial
state, s, the agent can choose to take an action $a \in A$. The state of the MDP
then randomly transitions to a successor state s'. The successor state is drawn
according to $s' \sim P_{sa}$. In MDPs, the transition model depends on the current
state, the next state, and the action of the agent. This process happens sequen-
tially over time, with each new action, generating a new successive state, which
leads to further actions.

The reward function, $R(s, a)$, is the primer driver of RL algorithms. Having
the correct $R(s, a)$ can make or break RL algorithms in that the reward function
is essentially the "teacher" in the learning algorithm. It indicates to the RL
algorithm which state-action pairs are more desirable than others by assigning
values. For example, one methodology is to assign positive value to state-action
pairs that are desirable and a zero or negative values to those that don't matter
or are not as desirable.

Thus, the goal in reinforcement learning is to maximize the reward over time,
choosing states via actions that increase the return over time, and avoiding states
that decrease the return over time. If we start at time t in state s_t and choose an
action a_t, where s_i and a_i are states and actions in a sequence, and $t = 0, 1, 2, \ldots,$
we can represent the the MDP as follows:

$$s_t \xrightarrow{a_t} s_{t+1} \xrightarrow{a_{t+1}} s_{t+2} \xrightarrow{a_{t+2}} s_{t+3} \cdots \tag{1}$$

Thus, the total reward at time t if we take $a_0, a_1, \ldots$ actions and visit $s_0, s_1, \ldots$ states over time $t+1, t+2, t+3, \ldots$ is given by:

$$R_t(s, a) = R_{t+1}(s_t, a_t) + \gamma R_{t+2}(s_{t+1}, a_{t+1}) + \gamma^2 R_{t+3}(s_{t+2}, a_{t+2}) + \cdots$$

$$R_t(s, a) = \sum_{i=1}^{\infty} \gamma^{i-1} R_{t+i}(s_{i-1}, a_{i-1}) \tag{2}$$

Thus, the goal would be to maximize the expected value of the total reward:

$$E[R_t(s, a)] \tag{3}$$

The solution to this problem is to find a *policy*, π, which returns the action that will yield the highest reward for each state. A policy is any function that maps states to actions: $\pi : S \mapsto A$. When we *execute* a policy π when in a state s, the agent takes action $a' = \pi(s, a)$. There can potentially be many policies to choose from, but only one can be considered an optimal policy, which is denoted by π^* and yields the highest expected reward over time, which is call the *action value function*. Thus, an action value function for a policy π is:

$$q^{\pi}(s, a) = E[R_t | S_t = s, A_t = a, \pi] \tag{4}$$

where $q^{\pi}(s, a)$ is the expected sum of discounted rewards when starting in state s and taking action a according to policy π.

The goal is to find an *optimal policy*, π^* that maximizes the action value function:

$$\pi^*_{(s,a)} = \operatorname*{argmax}_{\pi} q^{\pi}(s, a) \tag{5}$$

For most practical applications of RL, $q^{\pi}(s, a)$ is not known and must be learned by interacting with the environment. This type of RL agent, one that learns the action-value function not from a transition model for the environment, but rather from direct interaction with the environment, is called a model-free RL agent.

2.2 SARSA(λ)

In RL, rewards are viewed as short-term signals of the quality of an action, where as the action value function, $Q^{\pi}(s, a)$ represents the long-term value of a state-action pair. Temporal difference (TD) learning is a class of model-free methods that estimates Q^{π} as the agent interacts with the environment. The agent samples transitions and then updates the estimate of Q^{π} using observed and the estimate of the values of the next action. Typically, the agent makes these observations and updates the action value function at every time step, according to the following update rule:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \delta_t \tag{6}$$

where Q is an estimate of q^π, α is the step size, and δ_t is the TD error. SARSA is a TD learning algorithm that samples states and actions using an ϵ-greedy policy and then updates the Q values using Eq. 6 and a δ_t as follows:

$$\delta_t = R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t) \tag{7}$$

The term $R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})$ is called the target. It consists of the reward plus the discounted value of the next state and next action.

SARSA is known as an on-policy method which arises from the fact that the behavior policy u is the same as the target policy π. That is, the TD target in SARSA consists of $Q(S_{t+1}, A_{t+1})$, where A_{t+1} is sampled using μ. The target policy π, is used to compute the TD target. Although On-policy methods may result sub-optimal policies in certain instances, it has been shown that policies learned in on-policy methods tend to be safer when the risks are greater because SARSA takes the action selection into account [24].

Eligibility traces, a method of including information about not just the current time step, but information from multiple time steps, are a key mechanisms in reinforcement learning. For example, in TD(λ) algorithm, the λ refers to the use of an eligibility trace. Almost any TD method, such as SARSA, can be combined with eligibility traces to obtain a more general method that may learn more efficiently [24]. In this study, SARSA(λ) is used to improve efficiency of the learning agent.

2.3   Function Approximation Using Tile Coding

In order to implement Eqs. 6 and 7, an estimate of Q must be maintained and updated. Since there is no analytical way of expressing Q as a function, a method of funciton approximation is often used. Formally, function approximation is a technique for "representing the value function concisely at infinitely many points and generalizing value estimates to unseen regions of the state- action space" [23]. *Tile coding* is a linear function approximation method that is flexible and computationally efficient. In tile coding, the variable space, which is typically composed of states and actions, is partitioned into tiles, with each partition called a tiling. The method uses several overlapping tilings and for each tiling, maintains the weights of its tiles. The approximate value of an input is found by summing the weights of the tiles, one per tiling, in which it is contained. Given a training example, the method adjusts the weights of the involved tiles by the same amount to reduce the error on the example [23].

2.4   Action Selection Policy

As the action-value function is being learned via a function approximation method such as Tile Coding, the agent needs a method to select an action from the current set of actions whose action-value is known. Equation 5 always chooses the action that yields the largest action value. This is known as *greedy* action selection. But since the agent learns this function through experience, there may

be other actions that the agent may never see that will yield better values. In order to remedy this, three popular action selection strategies are often used: ϵ-greedy, ϵ-soft, and soft-max [24].

ϵ-greedy. In ϵ-greedy, the action with the highest value is chosen most of the time. However, every so often, an action is selected at random with ϵ probability. The action is selected uniformly, independent of the action values. This guarantees that every action will be explored sufficiently to find the optimal policy. Larger values of ϵ favor more exploration of actions, while smaller values favor exploitation of the greediest action. This is known as the exploration vs exploitation trade-off.

ϵ-soft. ϵ-soft action selection is similar to ϵ-greedy, except that the best action is selected with probability $1 - \epsilon$ and a random action is selected the rest of the time.

Softmax. A limitation of ϵ-greedy and ϵ-soft is when selecting random actions, they are selected uniformly. The worst action to take is selected with equal probability to the best action to take. Enter softmax. In softmax action selection, first, a rank or weight is assigned to each of the actions, according to their action-value estimate. Then, a random action is selected with regards to the weight associated with each action. This means that as the action-value function is learned, poor actions would be less likely to be chosen. This is useful, particularly the worst actions have unfavorable consequences.

For this study, an ϵ-greedy action selection strategy was used for its simplicity.

3   System Design and Experimental Setup

While having a working prosthetic to develop a solution for learning grip preferences using reinforcement is desirable, prosthetic devices are both expensive and challenging to configure and interface with. Thus, in order to carry out the agent design and experimentation in a low-cost way that would be accessible by other researchers or iML practioners, a simulation environment was created. The simulation consisted of a grip selection task with object grasping and was created using two popular and well supported open source projects, Pybullet and OpenAI gym.

3.1   Simulation Environment

The grip selection task was simulated using the Bullet physics simulator[1] and the OpenAI gym interface[2]. A robotic arm with three degrees of freedom was modeled and imported into the Bullet environment. The robotic arm consists of two grippers, each with one degree of freedom, and the arm portion with one degree of freedom. The arm is attached to a table on which objects of variable

---

[1] https://pybullet.org/wordpress/.
[2] https://gym.openai.com/.

size are placed, one at a time. Figure 1 shows different scenarios that are possible in the simulation.

Although the Bullet environment offers an API that can be used to control the environment to do things such as restart an episode, control the arm, or add an object of random size, the OpenAI gym interface provides a consistent, common API that can be used to develop agents that can operate in multiple environments, as opposed to developing environment specific agents. Thus, the OpenAI gym interface was used to abstract the specific environment details of the grasping environment away from the agent. This allows for de-coupling of the agent development from the environment development and allows other researchers to use the grasping environment with their own custom agents. By using the Bullet physics engine along with the OpenAI gym environment, several experiments were carried out with little effort.
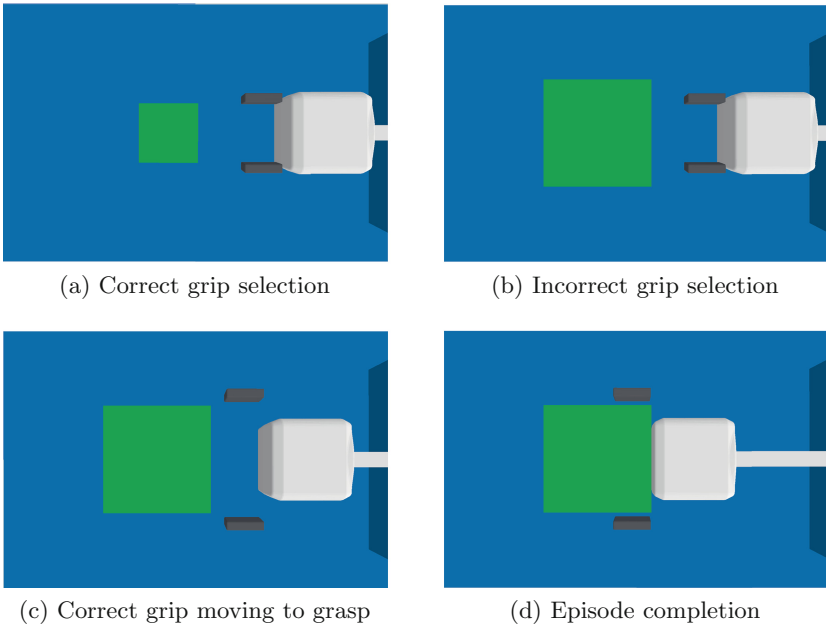

(a) Correct grip selection   (b) Incorrect grip selection

(c) Correct grip moving to grasp   (d) Episode completion

Fig. 1. Simulation environment in pybullet using the OpenAI gym interface. In (a), the arm is in the starting position called the "grip changing station". It has selected the correct grip for the object size. In (b), the incorrect grip has been selected, although the arm still remains in the grip chaning station. In (c), the correct grip for the object size as been enabled and the arm is moving forward to grasp the object. Finally, (d) illustrates the completion of an episode.

3.2 Spatial Interface Valuing

In order to have an agent learn from a human, some form of feedback is required. In this work, a learning agent perceives a value of its own behavior from human hand gestures given via a spatial interface, which I term Spatial Interface Valuing (SIV). In order to capture hand gestures, a Leap Motion Controller[3] (LMC) is used as the spatial interface device. The LMC provides tracking information for the left and right hands, as well as simple gesture recognition. The gestures of interest for this study are a thumbs up or thumbs down. Unfortunately, that is not one of the stock gestures implemented. Instead, the roll (ρ) of the right hand was used to determine whether the user was indicating a thumbs up or thumbs down. A simple piece-wise function was used to determine the state of the hand: thumbs up (1.0), thumbs down (−1.0).

$$hand\_state(\rho) = \begin{cases} 1.0 & \text{if } -45 < \rho < -135 \\ -1.0 & \text{otherwise} \end{cases} \tag{8}$$
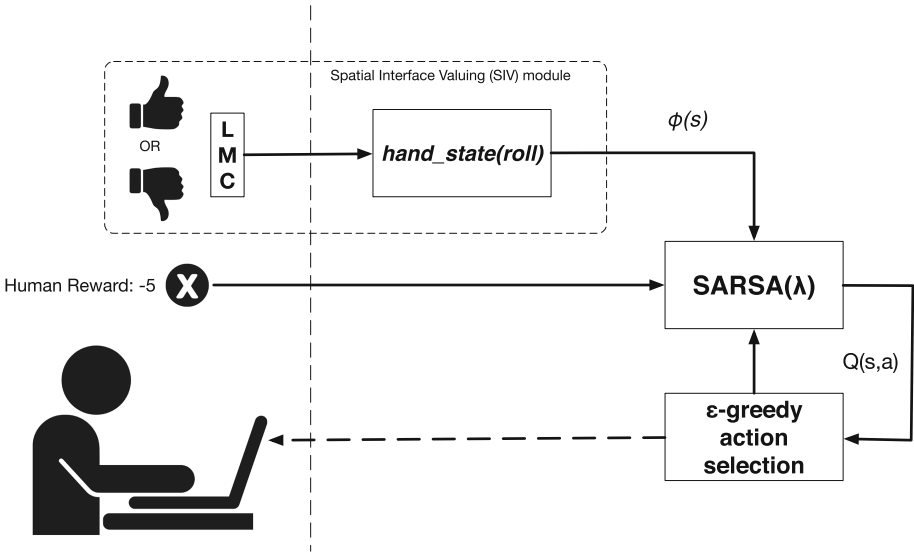

Fig. 2. Overview of the SIV agent setup

3.3 Experimental Setup

The experimental setup consists of a user observing the simulated grip-selection task and assisting the agent during the training process by (1) signaling a thumbs

---

up or thumbs down via the spatial interface to signal approval or disapproval of the agents behavior and (2) pushing the space bar on the keyboard when the agent was not behaving as expected, giving the agent a negative reward. A SARSA(λ) agent both with and without SIV, as well as baseline agent was evaluated, each for 3 runs with 15 episodes per run. All of the experiments were carried out without the user knowing which of the three agents was currently being evaluated, i.e. in a blind setting. Additionally, the order in which the experiments were carried out was randomly selected over the combination of runs, episodes, grips, object sizes, and agent. Figure 2 shows an overview of the experimental setup, with the only difference between agent the SIV/No SIV agents is the feedback from the hand state.

Note: Three distinct simulation environments were created for this study, a baseline environment for comparison, one environment with SIV feedback, and one environment without SIV. The experimental setup (excluding the SIV module) is the same, with the only difference being the inclusion (or exclusion) of the SIV module. For both simulation environments, the agent makes observations of the state space every *one-tenth of a second* and must take an action at every time step.

Table 1. State spaces for each agent under test: SIV vs no SIV

Agent type	State space variables
Baseline	current grip size, current object size, bias
With SIV	current grip size, hand state, bias
Without SIV	current grip size, bias

State Space. As mentioned above, three SARSA(λ) [22,24] agents, a baseline, one with SIV and one without SIV, were implemented to determine how well the agents learned a user's preferences for the grip selection task. Table 1 describes the state space design for all three agents. The SARSA(λ) baseline agent uses the size of the object and the current grip, along with a bias term as the state space vector $\phi(s)$:

$$\phi(s) = [current\_grip\_size, current\_object\_size, bias] \tag{9}$$

This state space was chosen in order to allow the agent to learn the best grip for each object, since both values are known form within the simulation, they can be observed and are used to establish a baseline level of performance with which to compare the SIV an No SIV enabled agents. The maximum grip size, maximum object size, number of grips, and number of objects are all parameters that can be configured prior to running each episode.

For the agent enabled with SIV, which is ideally for deployment in a real prosthetic device, the grip size is known to the agent, but for real-world grasping

tasks, object size is not known. This is where SIV steps in. Having knowledge of the user's gesture takes the place of exact knowledge of the object size (as in the baseline agent). Thus, the state vector for the SIV agent is:

$$\phi_{SIV}(s) = [current\_grip\_size, hand\_state, bias] \qquad (10)$$

And lastly, the state space for the agent that does not use SIV as a feedback channel, only has knowledge of the current grip.

$$\phi_{NoSIV}(s) = [current\_grip\_size, bias] \qquad (11)$$

Action Space. Modeled after the grip selection task in [26], the complete action space for the agents consists of the following:

$$\mathcal{A}_{complete}(s) = \{grip_1, grip_2, \ldots grip_n, \leftarrow, \rightarrow\} \qquad (12)$$

where the first $1 \ldots n$ actions are grip selections amongst the n grips. The remaining actions $\{\leftarrow, \rightarrow\}$, when taken, move the arm one step closer to the object ($\leftarrow$) or one step closer to the grip changing station ($\rightarrow$). However, the actions that are available to the agent depending on the position of the arm relative to the grip changing station, the object, and the reward. When the arm is in the grip changing station, the actions available are:

$$\mathcal{A}_{grip\_change}(s) = \{grip_1, grip_2, \ldots grip_n, \leftarrow\} \qquad (13)$$

Once the grip has been selected and the agent moves forward one step towards the object, the actions available are:

$$\mathcal{A}_{move}(s) = \{\leftarrow, \rightarrow\} \qquad (14)$$

And lastly, if the agent receives an explicit negative reward from the human user, the only available action is to return to the grip changing station:

$$\mathcal{A}_{return}(s) = \{\rightarrow\} \qquad (15)$$

Table 2. Hyperparameters for SARSA(λ) agents

| Parameter	Value
λ	0.0
step size	0.5
γ	1.0
ϵ	0.1

4   Experiments

4.1   Two Objects with Four Grips

In order to validate SIV as an RL method that can effectively learn a user's preference during grip selection, several trials of a grip selection task were carried out with a user in a blind setting, as mentioned in Sect. 3.3. At the beginning of each episode, the environments were setup to randomly select one of two object sizes: a small object, much smaller than the largest grip size, and a large object, approximately the size of the largest grip. The number of grips for the grip selection task was set to four for all episodes, requiring that the agent learn the users grip preference in two distinct grasping situations. The experiments were formulated as an episodic MDP with 0 discount ($\gamma = 1.0$), 0 reward at every time step and a 0 reward for completing the episode. The hyperameters for SARSA(λ) agents are show in Table 2.

4.2   Experimental Results

The results for the experiment are shown in Fig. 3. Overall, the baseline agent performs the best when considering the average steps per episode (Fig. 3(a)). This is to be expected, because once the baseline agent learns the user preference, it can execute those preferences directly because the baseline agent has knowledge of both the object size and current grip and has learned the action value function and thus, the optimal policy. Once trained, it does not need to rely on the human for any form of input. However, observing the performance of the agent using SIV, although on average it took more steps to complete an episode than the baseline, it out performed the agent without SIV on two of the three runs. This may be due to the human user themselves getting accustomed to providing the feedback to all of the agents.

Another interesting finding is that both the baseline agent and the agent with SIV learned with approximately the same number of average reward button pushes, where as the agent learning the grip selection task without SIV needed many more button pushes to successfully complete the runs. This indicates that the agent with SIV learns as quickly as the baseline agent, while the agent without SIV takes longer to learn the user's preference. This is likely because the agent without SIV only has the reward channel to try to learn the human users preference. This is shown in Fig. 3(b) and (e). Similarly, the amount of total reward needed and total number of button pushes needed (Fig. 3(d) and (e)) for the baseline and SIV agents is approximately half what is needed for the agent without SIV.
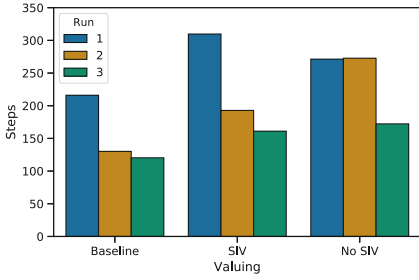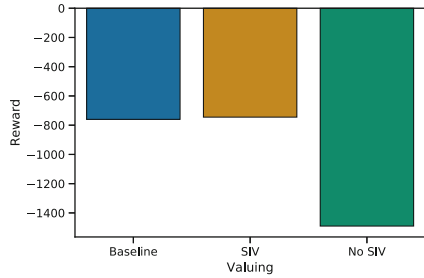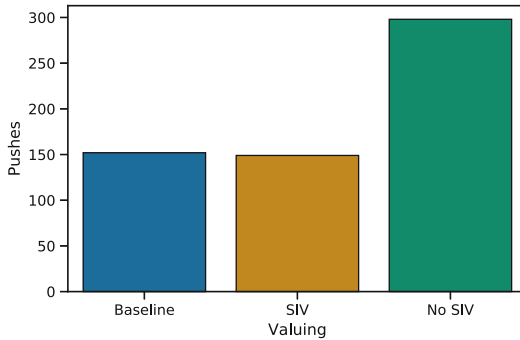
(a) Average steps per run
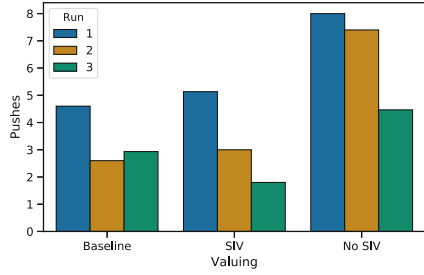
(b) Average pushes per run
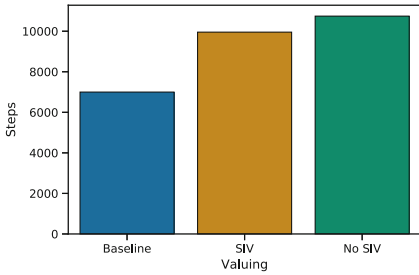
(c) Total steps per method

(d) Total reward per method

(e) Total pushes per method

Fig. 3. Results from evaluating agents using SIV against a Baseline agent with full knowledge of the current grip and object size, and an agent not using SIV with knowledge of only the current grip.

5   Conclusions and Discussion

This study introduces a new approach called *Spatial Interface Valuing* that uses a 3D spatial interface device to adapt an agent to a user's preferences. I showed that SIV can produce substantial performance improvements over an agent that does not use SIV. Through SIV, the agent learns to adapt its own performance

and learn user preferences through gestures, reducing the amount of explicit feedback required. SIV delivers implicit feedback from a user's hand gestures to an agent, allowing the agent to learn much more quickly and require less explicit human generated reward. The SIV agent learned to map hand gestures to user satisfaction, codifying satisfaction as a action value function using temporal methods for Reinforcement Learning. This technique is task agnostic and I believe it will easily extend to other settings, tasks, and forms of body language.

6   Future Work

The current results are encouraging, however, more work can be done on several fronts. This study was limited to studying the SARSA(λ) RL algorithm, an on-policy agent that uses tile coding, a function approximation technique, to estimate the action value function and learn a policy that encodes user preferences. It may be useful to study other types of agents such as off-policy agents using Q-learning, for example. Additionally, deep learning techniques such as Deep Q Networks [19] and DeepSARSA [27] may perform better. Additionally, simply extending the size of the experiment by conducting more runs with more users, different hyper-parameters and random seeds, and different object size/grip number combinations in a randomized control experiment will allow for a more robust statistical analysis of the performance.

References

1. Abbeel, P., Ng, A.Y.: Apprenticeship learning via inverse reinforcement learning. In: Proceedings of the Twenty-First International Conference on Machine Learning. ICML 2004. ACM, New York (2004)
2. Abel, D., Salvatier, J., Stuhlmüller, A., Evans, O.: Agent-agnostic human-in-the-loop reinforcement learning. In: NIPS Workshop on the Future of Interactive Learning Machines 2016, January 2017
3. Amodei, D., Olah, C., Steinhardt, J., Christiano, P., Schulman, J., Mané, D.: Concrete problems in AI safety. arXiv:1606.06565 [cs], June 2016
4. Boukhelifa, N., Bezerianos, A., Lutton, E.: Evaluation of interactive machine learning systems. arXiv:1801.07964 [cs], January 2018
5. Card, S.K., Moran, T.P., Newell, A.: The keystroke-level model for user performance time with interactive systems. Commun. ACM **23**(7), 396–410 (1980)
6. Card, S.K., Newell, A., Moran, T.P.: The Psychology of Human-Computer Interaction. L. Erlbaum Associates Inc., Hillsdale (1983)
7. Christiano, P., Leike, J., Brown, T.B., Martic, M., Legg, S., Amodei, D.: Deep reinforcement learning from human preferences. arXiv:1706.03741 [cs, stat], June 2017
8. Edwards, A.L., et al.: Application of real-time machine learning to myoelectric prosthesis control: a case series in adaptive switching. Prosthet. Orthot. Int. **40**(5), 573–581 (2016)
9. García, J., Fernández, F.: A comprehensive survey on safe reinforcement learning. J. Mach. Learn. Res. **16**, 1437–1480 (2015)

10. Hewett, T.T., et al.: ACM SIGCHI curricula for human-computer interaction. Technical report. ACM, New York (1992)
11. Holzinger, A.: Interactive machine learning for health informatics: when do we need the human-in-the-loop? Brain Inform. **3**(2), 119–131 (2016)
12. Knox, W.B., Setapen, A., Stone, P.: Reinforcement learning with human feedback in mountain car. In: AAAI Spring Symposium: Help Me Help You: Bridging the Gaps in Human-Agent Collaboration (2011)
13. Knox, W.B., Stone, P.: Augmenting reinforcement learning with human feedback. In: ICML Workshop on New Developments in Imitation Learning, p. 8 (2011)
14. Knox, W.B., Stone, P.: Framing reinforcement learning from human reward. Artif. Intell. **225**(C), 24–50 (2015)
15. Knox, W.B., Stone, P., Breazeal, C.: Teaching agents with human feedback: a demonstration of the TAMER framework. In: Proceedings of the Companion Publication of the 2013 International Conference on Intelligent User Interfaces Companion, pp. 65–66 Companion. ACM, New York (2013)
16. Li, C., Ren, J., Huang, H., Wang, B., Zhu, Y., Hu, H.: PCA and deep learning based myoelectric grasping control of a prosthetic hand. BioMedical Eng. OnLine **17**, 107 (2018)
17. Liu, F., Su, J.B.: Reinforcement learning based on human-computer interaction. In: Proceedings. International Conference on Machine Learning and Cybernetics, vol. 2, pp. 623–627, November 2002
18. Mathewson, K., Pilarski, P.M.: Simultaneous control and human feedback in the training of a robotic agent with actor-critic reinforcement learning. In: Interactive Machine Learning Workshop at IJCAI 2016, p. 7 (2016)
19. Mnih, V., et al.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (2015)
20. Ng, A.Y., Russell, S.J.: algorithms for inverse reinforcement learning. In: Proceedings of the Seventeenth International Conference on Machine Learning, ICML 2000, pp. 663–670. Morgan Kaufmann Publishers Inc., San Francisco (2000)
21. Resnik, L., Meucci, M.R., Lieberman-Klinger, S., Fantini, C., Kelty, D.L., Disla, R., Sasson, N.: Advanced upper limb prosthetic devices: implications for upper limb prosthetic rehabilitation. Arch. Phys. Med. Rehabil. **93**(4), 710–717 (2012)
22. Rummery, G.A., Niranjan, M.: On-line Q-learning using connectionist systems. Cambridge University Engineering Department, Technical report (1994)
23. Sherstov, A.A., Stone, P.: Function approximation via tile coding: automating parameter choice. In: Zucker, J.-D., Saitta, L. (eds.) SARA 2005. LNCS (LNAI), vol. 3607, pp. 194–205. Springer, Heidelberg (2005). https://doi.org/10.1007/11527862_14
24. Sutton, R.S., Barto, A.G., Bach, F.: Reinforcement Learning: An Introduction, 2nd edn. A Bradford Book, Cambridge (2018)
25. Thomaz, A.L., Breazeal, C.: Teachable robots: understanding human teaching behavior to build more effective robot learners. Artif. Intell. **172**(6–7), 716–737 (2008)
26. Veeriah, V., Pilarski, P.M., Sutton, R.S.: Face valuing: training user interfaces with facial expressions and reinforcement learning. CoRR abs/1606.02807 (2016)
27. Zhao, D., Wang, H., Shao, K., Zhu, Y.: Deep reinforcement learning with experience replay based on SARSA. In: 2016 IEEE Symposium Series on Computational Intelligence (SSCI), pp. 1–6, Decemner 2016